Backpropagation

Vorlesung 2, Deep Learning

Dozenten: Prof. Dr. M. O. Franz, Prof. Dr. O. Dürr

HTWG Konstanz, Fakultät für Informatik

Übersicht

Übersicht

1. Gradientenabstieg für neuronale Netze

2. Backpropagation

3. Beispieltraining eines MLPs

Wiederholung: Matrixnotation für MLPs

Die Aktivierungen a_j^l der Schicht l werden in einem **Aktivierungsvektor** a^l zusammengefasst, genauso die Schwellwerte a_j^l in einem **Schwellwertvektor** b^l. Die Gewichte für das j-te Neuron bilden die j-te Reihe der **Gewichtsmatrix** w^l.

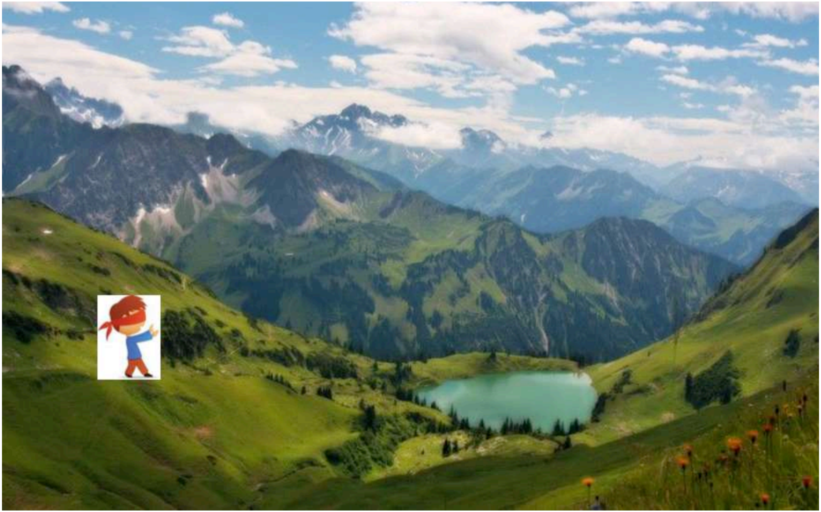
Unter der Annahme, dass $\sigma(z)$ *vektorisiert* ist (d.h. auf jede Komponente des Eingangsvektors angewandt wird), ergibt sich die vereinfachte Matrixschreibweise:

$$\boxed{a^l = \sigma(w^l a^{l-1} + b^l).}$$

Die Zwischengröße $z^l = w^l a^{l-1} + b^l$ wird oft gebraucht und bezeichnet den **gewichteten Input**.

Trainingsaufgabe (MLP): finde geeignete Gewichtsmatritzen w^l und Schwellwertvektoren b^l für jede Schicht l.

Wiederholung: Gradientenabstieg (1)



Hier: zweidimensionale Kostenfunktion. Bei neuronalen Netzen haben wir soviele Dimensionen wie die Anzahl der Koeffizienten in den Gewichtsmatritzen und Schwellwertvektoren!

Wiederholung: Gradientenabstieg (2)

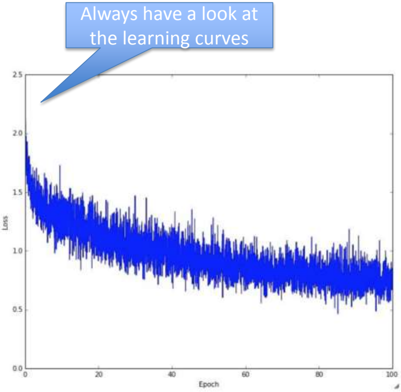
Gradientenabstieg

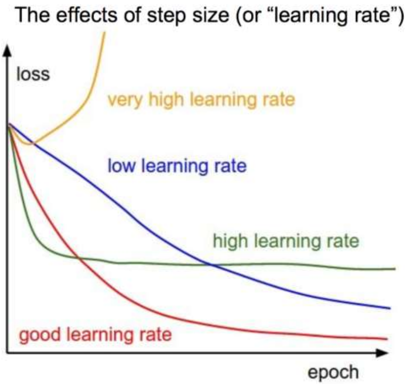
begin initialize $w^l, b^l, \theta, \eta_k, k = 0$
    do $k \leftarrow k + 1$
        $w^l \leftarrow w^l - \eta_k \nabla_w C$
        $b^l \leftarrow b^l - \eta_k \nabla_b C$
    until $\eta_k(\|\nabla_a C\| + \|\nabla_b C\|) < \theta$
    return w^l, b^l
end

Probleme:

- Gradientenabstieg findet nur lokale Minima.
- Wenn η_k zu klein gewählt wird, ist die Konvergenzrate zu klein, oder die Prozedur konvergiert gegen einen Wert, der kein Minimum ist.
- Wenn η_k zu groß gewählt wird, kann das Ziel verfehlt werden, oder die Prozedur oszilliert zwischen 2 Werten, oder divergiert sogar.

Lernkurven und Epochen





Lernkurven: Leistungsparameter einer Lernmaschine (meist Fehler oder Kostenfunktion), aufgetragen über Durchläufe durch den Datensatz (sog. **Epochen**).

Kostenfunktionen für neuronale Netze

Neuronale Netze haben oft Millionen von Gewichten. Für eine effektive Berechnung des Gradienten muss die Kostenfunktion zwei Bedingungen erfüllen:

1. Die Gesamtkosten für einen Trainingsdatensatz ergeben sich als Summe der Einzelkosten der Trainingsbeispiele x_k:

   $$C(x_1, x_2, \ldots) = \frac{1}{n} \sum_k C_{x_k} \quad \text{Beispiel:} \quad C = \frac{1}{2n} \sum_k (y_k - a_k^L)^2.$$

   Damit gilt $\nabla C = \sum_k \nabla C_{x_k}$, d.h. der Gradient ist der **Durchschnitt der Gradienten der einzelnen Trainingsbeispiele**.

2. Die Kostenfunktion darf nur von den Aktivierungen a_k^L der letzten Schicht abhängen. Dies ermöglicht eine schichtenweise Berechnung des Gradienten, ausgehend von der letzten Schicht (**Backpropagation**).

Stochastischer Gradientenabstieg

- Die Kostenfunktion für einen Schritt des Gradientenabstiegs muss über den gesamten Trainingsdatensatz berechnet werden. Bei den im Deep Learning üblichen Trainingssetgrößen im Bereich $10^4 - 10^9$ würde es daher sehr lange dauern, bis man überhaupt einen Verbesserungsschritt machen kann.
- Statt den durchschnittlichen Gradienten $\nabla C = \frac{1}{n} \sum_k \nabla C_{x_k}$ über den gesamten Trainingsdatensatz zu berechnen, kann man auch ein zufällig gewählte Untermenge $\mathcal{M}$ der Trainingsdaten auswählen und damit eine näherungsweise Schätzung des Gradienten erhalten. Diese Untermenge heißt **Minibatch**.
- Ist der Minibatch groß genug, dann gilt

$$\frac{1}{m} \sum_{k \in \mathcal{M}} \nabla C_{x_k} \approx \frac{1}{n} \sum_k \nabla C_{x_k}.$$

- Ergänzt man den Gradientenabstieg um diesen Schritt, so erhält man einen **stochastischen Gradientenabstieg** mit deutlich häufigeren Updates und schnellerer Konvergenz.

Übersicht

Kettenregel

Die Ableitungen von verschachtelten Funktionen

$$f(x) = a\left(b(x)\right) \quad \text{bzw.} \quad f(x) = a\left(b_1(x), b_2(x), \ldots\right)$$

können durch Multiplikation der einzelnen Ableitungen entlang der Aufrufstruktur gebildet werden:

$$\frac{\partial f}{\partial x} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial x} \quad \text{bzw.} \quad \frac{\partial f}{\partial x} = \sum_k \frac{\partial a}{\partial b_k} \cdot \frac{\partial b_k}{\partial x}$$

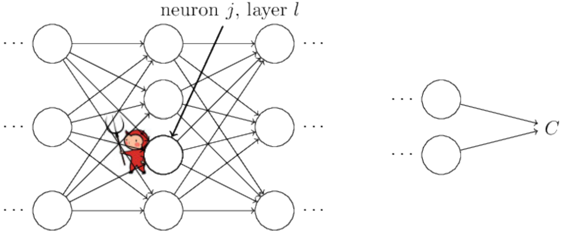
Neuronale Netze sind "Stapel" aus Neuronenschichten, jede beschrieben durch eine multivariate Funktion. Der Netzoutput ist eine verschachtelte Funktion, gebildet durch Verkettung dieser Funktionen, z.B.

$$f(x) = a\left(b\left(c\left(d\left(e(x)\right)\right)\right)\right) \quad \text{bzw.} \quad f = a \circ b \circ c \circ d \circ e$$

mit der Ableitung

$$\frac{\partial f}{\partial x} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial c} \cdot \frac{\partial c}{\partial d} \cdot \frac{\partial d}{\partial e} \cdot \frac{\partial e}{\partial x}.$$

Fehler eines Neurons bzw. einer Schicht



[Nielsen, 2016]

Eine kleine Änderung Δz_j^l des gewichteten Inputs an Neuron Nr. j in Schicht l führt zu einer Veränderung der Kostenfunktion um $\frac{\partial C}{\partial z_j^l}\Delta z_j^l$. Wenn $\frac{\partial C}{\partial z_j^l}$ nahe an 0 ist, kann die Kostenfunktion durch Δz_j^l nicht verbessert werden. Daher nennt man den Term

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

den **Fehler** des Neurons bzw. δ^l den **Fehlervektor** von Schicht l.

Backpropagation (Idee)

- Die Kostenfunktion hängt über eine Verkettung von den gewichteten Inputs nur über deren Nachfolgeschicht ab:

$$C = C(z^{l+1}(z^l)) \quad \text{bzw.} \quad C \circ \cdots \circ z^{l+1} \circ z^l \circ \cdots$$

- Partielle Ableitungen (in unserem Fall die Fehler jeder Schicht) können daher beginnend an der Zielfunktion "rückwärts" durch das neuronale Netz propagiert werden: es gilt

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l}.$$

- Laufzeit: Günstig, da Teillösungen (bereits bekannte partielle Ableitungen) wiederverwendet werden können. Ableitungen der einzelnen Funktionen können oft in konstanter Zeit berechnet werden.

1. Fundamentalgleichung

Der Fehler in der Ausgangsschicht L ist $\delta_j^L = \frac{\partial C}{\partial z_j^L}$. Die Kostenfunktion für ein fixes Input-Label-Paar hängt nur von den Aktivierungen a_j^L der Ausgangsschicht ab, daher gilt nach der Kettenregel:

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L} \cdot \frac{\partial a_k^L}{\partial z_j^L}.$$

Der Output von Neuron j hängt nur von seinem eigenen gewichteten Input z_j^L ab, daher verschwinden alle anderen Summenterme für $k \neq j$:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L}.$$

Wegen $a_j^L = \sigma(z_j^L)$ ergibt sich die erste Fundamentalgleichung

$$\boxed{\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L).}$$

2. Fundamentalgleichung

Fehler an einem beliebigen Neuron ($l < L$, wieder Kettenregel):

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}.$$

Die Ableitung von $z_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_j^{l+1}$ ist

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l).$$

Einsetzen ergibt die 2. Fundamentalgleichung:

$$\boxed{\delta_j^l = \sum_k w_{kj}^{l+1} \sigma'(z_j^l) \delta_k^{l+1}.}$$

Berechnet den Fehler in Schicht l aus dem Fehler in Schicht $l + 1$.

3. Fundamentalgleichung

Ableitung der Kostenfunktion nach dem Gewicht (Kettenregel):

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_j} \cdot \frac{\partial z^l_j}{\partial w^l_{jk}} = \frac{\partial z^l_j}{\partial w^l_{jk}} \delta^l_j.$$

Die Ableitung von $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$ nach w^l_{jk} ist

$$\frac{\partial z^l_j}{\partial w^l_{jk}} = a^{l-1}_k.$$

Einsetzen ergibt die 3. Fundamentalgleichung:

$$\boxed{\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j.}$$

Damit berechnet sich das Update der Gewichte aus dem Fehler der gleichen Schicht.

4. Fundamentalgleichung

Ableitung der Kostenfunktion nach dem Schwellwert (Kettenregel):

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial z_j^l}{\partial b_j^l} \delta_j^l.$$

Die Ableitung von $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$ nach b_j^l ist

$$\frac{\partial z_j^l}{\partial b_j^l} = 1.$$

Einsetzen ergibt die 4. Fundamentalgleichung:

$$\boxed{\frac{\partial C}{\partial b_j^l} = \delta_j^l.}$$

Damit berechnet sich auch das Update des Schwellwerts aus dem Fehler der gleichen Schicht.

Fundamentalgleichungen in Matrixform

$$\begin{aligned}
&1. \quad & \delta^L &= \nabla_{a^L} C \odot \sigma'(z^L) \\
&2. \quad & \delta^l &= \left((w^{l+1})^\top \delta^{l+1} \right) \odot \sigma'(z^l) \\
&3. \quad & \nabla_{w^l} C &= \delta^l (a^{l-1})^\top \\
&4. \quad & \nabla_{b^l} C &= \delta^l
\end{aligned}$$

$\odot$ ist das **Hadamardprodukt**, d.h. eine elementweise Multiplikation von Vektoren oder Matritzen.

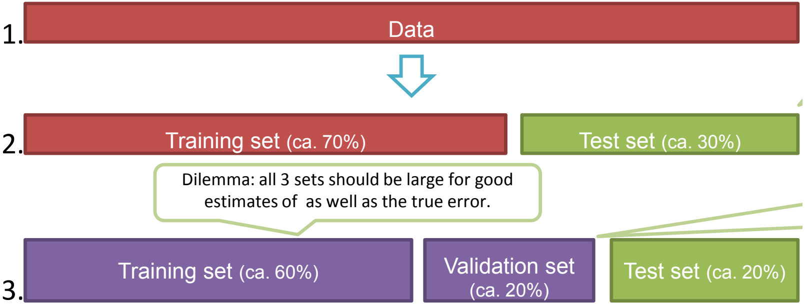
Backpropagation-Algorithmus

Backpropagation

Für den Trainingsinput x und -output y:

1. **Input:** Setze die Eingangsneuronen $a^1 = x$.
2. **Vorwärtslauf:** Berechne für alle $l = 2, 3, \ldots$ die gewichteten Inputs $z^l = w^l a^{l-1} + b^l$ und die Aktivierungen $a^l = \sigma(z^l)$.
3. **Fehler am Output:** Berechne den Vektor $\delta^L = \nabla_{a^L} C \odot \sigma'(z^L)$.
4. **Rückwärtslauf:** Berechne für alle $l = L-1, L-2, \ldots, 2$ den Fehler $\delta^l = \left((w^{l+1})^\top \delta^{l+1}\right) \odot \sigma'(z^l)$.
5. **Update:** Die Ableitungen der Kostenfunktion ergeben sich aus $\nabla_{w^l} C = \delta^l (a^{l-1})^\top$ und $\nabla_{b^l} C = \delta^l$.

Aufteilung der Daten für das Training neuronaler Netze



Validierungsdatensatz: verwendet zur Überprüfung des Lernfortschritts (z.B. über Lernkurven) und zur Schätzung von Hyperparametern. Ein volle Kreuzvalidierung ist meist zu aufwendig.

Übersicht

Training eines MLPs für MNIST mit Backpropagation

- Kostenfunktion: **mittlerer quadratischer Fehler** (MSE)

$$C(a^L) = \tfrac{1}{2} \sum_k (y_k - a_k^L)^2$$

mit dem Gradienten

$$\nabla_{a^L} C = a^L - y.$$

- Aktivierungsfunktion: Sigmoid

$$\sigma(z^l) = \frac{1}{1 + e^{-z^l}}$$

mit der Ableitung

$$\sigma'(z^l) = \sigma(z^l) \cdot (1 - \sigma(z^l)).$$

[Demo]